https://doi.org/10.31891/2219-9365-2025-84-5
УДК 004.056:004.852:004.75

ЛИГУН Олексій
Хмельницький національний університет
https://orcid.org/0009-0004-5727-5096
e-mail: oleksii.lyhun@gmail.com

МЕТОД ВИЯВЛЕННЯ ЗЛОВМИСНИХ ДРОПЕРІВ НА ОСНОВІ ГРАФОВОЇ УВАГИ ТА МОДЕЛЕЙ ВИКЛИКІВ API

У роботі представлено вдосконалений метод виявлення зловмисних дроперів у комп'ютерних системах, який ґрунтується на побудові орієнтованих графів викликів API та їх подальшому аналізі за допомогою Graph Attention Networks (GAT). Запропонований підхід орієнтований на сучасні типи дроперів, що використовують поліморфізм, метаморфізм, різноманітні техніки обфускації, динамічне завантаження компонентів та умовне виконання коду, які ускладнюють їх виявлення традиційними засобами.

На відміну від сигнатурних методів, що залежать від наявності відомих зразків шкідливих програм, та евристичних підходів, які часто страждають від високої кількості хибних спрацювань, модель на основі GAT дозволяє аналізувати структурні та контекстні залежності між викликами API. Завдяки механізмам уваги мережа здатна визначати найбільш інформативні вершини і ребра в графі, що відображають приховані патерни поведінки дроперів, незалежно від модифікацій їхнього машинного коду.

У роботі виконано детальний експериментальний аналіз із використанням корпусу Windows PE-файлів, що включає як легітимні, так і шкідливі зразки різних сімейств. Метод порівняно з базовими моделями машинного навчання, такими як Random Forest, SVM та градієнтний бустинг. Отримані результати демонструють суттєві переваги підходу на основі GAT як у точності класифікації, так і у стійкості до складних технік обфускації, що підтверджує його ефективність для практичного застосування в системах захисту.

Ключові слова: комп'ютерні системи, дропер, Graph Attention Network, виклики API, графова модель, обфускація, виявлення шкідливого ПЗ.

LYHUN Oleksii
Khmelnytskyi National University

A METHOD FOR DETECTING MALICIOUS DROPPERS BASED ON GRAPH ATTENTION AND API CALL PATTERNS

The paper presents an improved method for detecting malicious droppers in computer systems, which is based on the construction of directed graphs of API calls and their further analysis using Graph Attention Networks (GAT). The proposed approach is focused on modern types of droppers that use polymorphism, metamorphism, various obfuscation techniques, dynamic loading of components and conditional code execution, which complicate their detection by traditional means.

Unlike signature methods, which depend on the presence of known malware samples, and heuristic approaches, which often suffer from a high number of false positives, the GAT-based model allows for the analysis of structural and contextual dependencies between API calls. Thanks to the mechanisms of attention, the network is able to determine the most informative vertices and edges in the graph, which reflect the hidden patterns of droppers' behavior, regardless of modifications to their machine code.

In the work, a detailed experimental analysis is performed using a corpus of Windows PE files, which includes both legitimate and malicious samples of various families. The method compared to basic machine learning models such as Random Forest, SVM and gradient boosting. The obtained results demonstrate significant advantages of the GAT-based approach in both classification accuracy and resistance to complex obfuscation techniques, which confirms its effectiveness for practical application in security systems.

Keywords: computer, dropper, Graph Attention Network, API calls, graph-based model, obfuscation, malware detection

Стаття надійшла до редакції / Received 02.09.2025
Прийнята до друку / Accepted 22.10.2025

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Сучасні шкідливі програми все частіше будуються за багатоетапною структурою, де на початковій фазі застосовується спеціалізований компонент доставки — дропер. Цей клас ПЗ виконує завантаження, розпакування, шифрування або ін'єкцію основного шкідливого навантаження в систему жертви [1], [2]. Еволюція дроперів пов'язана із застосуванням метаморфізму, умовного виконання, відкладеної активації, а також аналітичних анти засобів, що дозволяє їм ухилятись від виявлення в пісочницях або у середовищах автоматичного аналізу [3], [4].

Зафіксовано численні приклади використання дроперів у реальних атаках із застосуванням Cobalt Strike, Emotet, SmokeLoader, які показали високу стійкість до сигнатурного виявлення навіть за умов ретельної поведінкової перевірки [5], [6]. У зв'язку з цим традиційні системи виявлення, зокрема антивіруси на основі сигнатур, а також прості евристичні правила, часто виявляються неефективними [7]. Просте виявлення на основі набору статичних ознак, ентропійного аналізу або кількості секцій у PE-файлі [8], [9] не дозволяє

коректно класифікувати дропери, які динамічно модифікують свій вміст або діють через віддалені сервери керування (C2).

У відповідь на це зростає інтерес до структурно-поведінкових методів виявлення, які аналізують сліди активності програми у вигляді викликів системних функцій (API), взаємодії з реєстром, файловою системою, мережею [10]. З точки зору машинного навчання, ці виклики можна подати у вигляді орієнтованого графа, де вузли — конкретні функції (наприклад, CreateProcess, RegSetValueEx), а ребра — їх послідовність [11]. Такий граф відображає внутрішню логіку шкідливої програми й може бути використаний для її ідентифікації незалежно від конкретного шляху зараження чи упаковки.

Однак класифікація таких графів потребує складніших методів, ніж класичні MLP або Random Forest, оскільки останні потребують ручного виокремлення ознак і не зберігають топологію зв'язків [12]. Тому у фокусі останніх досліджень — графові нейронні мережі (GNN), зокрема Graph Attention Network (GAT), яка дозволяє зважувати зв'язки між елементами поведінки в залежності від їх значущості [7], [9].

У роботах [1], [2] представлено розподілені архітектури виявлення з пастками й наживками, які можна адаптувати до потокового виявлення API-викликів у реальному часі. Статті [3], [4], [5] демонструють, що навіть при високому рівні обфускації мережеві або API-виклики залишають характерні шаблони — достатні для виявлення за умови правильного представлення.

Додаткові дослідження підтверджують ефективність глибокого навчання у задачах класифікації шкідливих програм. Наприклад, застосування згорткових мереж до зображень байт-коду [19], а також autoencoder-архітектур [27] показує надійні результати. Інструменти, такі як AVClass [20] та Drebin [24], стали де-факто стандартами в автоматизованій класифікації родин шкідливого ПЗ.

Огляд [22] узагальнює понад 50 методів застосування машинного навчання у боротьбі зі шкідливими файлами, тоді як [28] виділяє нові виклики в області explainable AI, adversarial атак і потокового виявлення. Значний прогрес відзначається в аналізі графів поведінки з використанням таких систем, як MalFox [18], які демонструють надійне виявлення завдяки інтеграції з графовими представленнями API. Розробки у сфері безпечного ML також включають методи захисту від атак типу adversarial examples [17], [21].

У сфері реверс-інжинірингу й аналізу виконання активне застосування отримали платформи Ghidra [25] і MalwareBazaar [26], що дозволяє збирати великі набори для навчання моделей. Водночас, аналіз графів активності в масштабі інтернету дозволив створити моделі на базі великомасштабного графового навчання [23], що ще більше розширює горизонти виявлення.

Таким чином, потреба у методах аналізу поведінки дроперів на основі графових моделей є актуальною як з наукової точки зору, так і з точки зору практичної реалізації в інструментах кіберзахисту [13].

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Існуючі методи виявлення шкідливого ПЗ поділяються на три основні категорії: сигнатурні, евристичні та поведінкові. У випадку з дроперами, які використовують шифрування та упаковку, сигнатурні методи виявляються малоефективними [3], [4]. Евристичні підходи, зокрема аналіз структури PE-файлів або ентропії секцій [8], [9], дещо покращують результативність, але залишаються вразливими до комбінованих технік ухилення.

Сучасні роботи фокусуються на застосуванні машинного навчання (ML) [14] і глибокого навчання (DL) для виявлення аномалій у поведінці програм або системних викликах. Наприклад, автори [16] використали модель CNN для обробки зображень, отриманих із байт-коду. Дослідження [15] запропонувало векторизацію послідовностей викликів API з подальшою класифікацією за допомогою LSTM. Альтернативно застосовуються autoencoder-архітектури, які дозволяють стискати поведінкові шаблони в компактні латентні простори [29], а також трансформери для послідовностей API-викликів [30].

Деякі автори розглядають attention-моделі з обмеженою пам'яттю (наприклад, Mamba), які можуть забезпечити кращу продуктивність на великих логах із поведінкових журналів [31]. Модифіковані CNN, як-от Attention-Guided CNN [32], також продемонстрували переваги при роботі зі структурованими даними, включаючи API-послідовності.

В останні роки активізувалося використання графових нейронних мереж (GNN), зокрема Graph Attention Networks (GAT), які показують високу точність у задачах класифікації вузлів і графів, а також здатні виявляти приховані шаблони взаємодії між компонентами [7], [13]. GAT дозволяє зосередити обчислювальну увагу на важливих вузлах у графі, що є критичним при аналізі викликів API зловмисного ПЗ. Автори огляду [15] відзначають застосування GAT у кібербезпеці як одну з найбільш перспективних тенденцій на найближчі роки. Також досліджуються альтернативні архітектури — Graph Convolutional Networks (GCN), GraphSAGE, DiffPool, Graph Transformers — які забезпечують гнучкі стратегії агрегації та масштабування [33]–[35].

З точки зору структурної ролі, дропер виконує функцію першого етапу атаки — забезпечує потрапляння основного шкідливого коду в систему жертви, його розгортання, активацію і маскування [41], [42]. Його архітектура зазвичай включає модуль перевірки середовища, модуль доставки (dropper core), декриптор або розпакувальник, а також засоби приховування активності, як-от використання API VirtualAlloc, CreateRemoteThread, реєстрових ключів Run, та навіть утиліти Windows Management Instrumentation (WMI)

[43]. Це робить дропери складними для виявлення, оскільки навіть у видозміненому вигляді вони повторюють характерну поведінкову модель — що якраз і дозволяє застосовувати до них графовий аналіз та GNN-моделі.

Якість програмного забезпечення (ПЗ) є багатовимірною характеристикою, яка визначає здатність програмного продукту ефективно виконувати свої функції, бути зручним у підтримці, безпечним, продуктивним і прозорим у структурі. У контексті виявлення зловмисного програмного забезпечення, зокрема дроперів, поняття якості набуває додаткового значення, оскільки дозволяє виявляти структурні дефекти, які можуть сигналізувати про приховану шкідливу активність або вбудовані модулі з ризиками безпеки.

Сучасна практика оцінювання якості ПЗ базується на використанні структурних метрик, що дозволяють кількісно описати складність, взаємозв'язаність, спадковість та підтримуваність коду [10]. Серед найбільш поширених показників: кількість рядків коду (LOC), циклічна складність (CC), глибина спадкування (DIT), відповідь класу (RFC), вага методів класу (WMC), а також показники дублювання, кількість дочірніх або батьківських класів (NOC/NOP). Наприклад, надмірна кількість LOC або високе значення циклічної складності свідчать про перевантаження окремих методів або класів, що ускладнює їхню перевірку, рефакторинг і підвищує ймовірність прихованих помилок [12], [14]. Додатково, високий рівень RFC або глибина спадкування DIT свідчать про складність архітектурної взаємодії та потенційно низьку інкапсуляцію.

Окрім базових метрик, стандарти ISO/IEC 25010 та SQALE вводять інтегральні показники якості, які поєднують вимірювані характеристики з аналітичними вагами, враховуючи контекст використання ПЗ [36], [37]. Наприклад, індекс підтримуваності (Maintainability Index), використовуваний у SonarQube, є одним з найпоширеніших у сучасних системах оцінки [38].

У науковій літературі запропоновано ряд інтегральних метрик, які об'єднують кілька показників у єдиний формалізований критерій оцінки. Одним із таких є показник структурної ризикованості методу (1):

$$\text{quality}_{\text{method}} = \frac{1000 \cdot CC + 100 \cdot DIT + 20 \cdot RFC + 15 \cdot NOP}{LOC} \quad (1)$$

Модифікації цієї формули з урахуванням додаткових показників (наприклад, глибини вкладеності циклів, або середнього числа параметрів) обговорено в роботах [39], [40], які пропонують адаптацію метрик якості під задачі аналізу кібербезпеки.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

У типовій структурі сучасної багатоетапної атаки дропер відіграє ключову роль як стартовий компонент, який виконує первинну інфекцію системи. Його завдання — непомітно доставити основне шкідливе навантаження, встановити контроль над середовищем виконання та, за потреби, завантажити інші модулі. Типовий ланцюг атаки включає такі етапи: отримання дропера через фішинг або експлойт, його запуск у системі, декодування або завантаження наступного компонента, ін'єкція в процес або розпакування, встановлення з'єднання з C2-сервером, отримання payload або виконання цільового шкідливого коду. В залежності від реалізації, дропер може бути автономним (включає все необхідне) або "тонким" — тобто діяти лише як downloader. Такі архітектури роблять неможливим просте сигнатурне виявлення й вимагають переходу до поведінкових моделей аналізу.

Запропонований метод виявлення дроперів базується на поведінковому аналізі програмного забезпечення у вигляді орієнтованого графа викликів API (Directed API Call Graph, DACG) та подальшому використанні графової нейронної мережі типу Graph Attention Network (GAT) для класифікації поведінкових шаблонів [7]. Основна ідея полягає в тому, що незалежно від обфускації, упаковки чи специфіки реалізації, дропери мають відносно стабільні структури послідовностей дій — вони ініціалізують специфічні API, готують пам'ять, взаємодіють з диском і реєстром, створюють нові процеси або завантажують шкідливі бібліотеки. Саме ці дії в сукупності можуть бути представлені у вигляді графа, де вузли — це виклики функцій API, а ребра — послідовність або контекстна залежність між ними.

Для формування такого графа виконання, кожен досліджуваний зразок (наприклад, PE-файл для Windows) запускається в контрольованому середовищі (sandbox), де здійснюється динамічне трасування системних викликів за допомогою моніторингових інструментів (Process Monitor, API Monitor, Sysmon або спеціалізованих модулів Any.Run, Cuckoo Sandbox). Журнал дій представляється у вигляді послідовності подій: час — назва процесу — API — аргументи — результат. Із цієї послідовності будується граф, де кожному унікальному API відповідає вузол, а зв'язки між викликами встановлюються за часовою послідовністю або логічною залежністю. (наприклад, виклик LoadLibrary → GetProcAddress утворює ребро, як і CreateFile → WriteFile). В табл. 1 подано характерні API-послідовності, що використовуються для побудови поведінкових графів, які є основою для навчання моделі GAT.

Для кожного вузла формуються ознаки, які включають такі категорії: ідентифікатор API (однокодове кодування або embedding), частота виклику, контекст виконання (в якому процесі і з якими параметрами), тип доступу (читання, запис, виконання), категорія дії (мережева, файлово-системна, процесна, реєстрова тощо). Такий вектор ознак дозволяє відобразити не лише факт виклику, а і його функціональне значення в контексті загального виконання.

Таблиця 1

Приклади API-ланцюжків, що формують ознаки графа поведінки

Example #	API Sequence	Behavioral Category
1	VirtualAlloc → WriteProcessMemory → CreateRemoteThread	Code Injection
2	LoadLibrary → GetProcAddress → CallFunction	Dynamic Linking
3	CreateFile → WriteFile → CloseHandle	Filesystem Activity
4	RegCreateKeyEx → RegSetValueEx → RegCloseKey	Registry Modification

Отриманий граф конвертується у формат, придатний для обробки GNN-моделлю — зазвичай у вигляді списку суміжності та матриці ознак вузлів. Архітектура моделі передбачає використання двох графових шарів GAT, які реалізують обчислення ваг зв'язків (коефіцієнтів уваги) для кожної пари суміжних вузлів. Завдяки цьому модель може зосередитись на найбільш значущих шляхах, які характерні саме для зловмисної поведінки. Наприклад, наявність послідовностей типу VirtualAlloc → WriteProcessMemory → CreateRemoteThread з високими вагами уваги прямо свідчить про намагання впровадити shellcode у сторонній процес, що є типовою технікою дропера [10].

Після побудови графа DACG для кожного зразка дані подаються до моделі графової нейронної мережі (Graph Attention Network), яка реалізує багатошарове перетворення ознак вузлів із урахуванням структурної топології та вагових коефіцієнтів між зв'язаними API-викликами.

На етапі навчання модель отримує матрицю ознак вузлів $X \in R^{n \times d}$, де n – кількість API викликів, d – розмірність ознак (embedding, категорія дії, частота, PID тощо), список суміжності або індекси ребер $E = \{\{i, j\}\}$ – для побудови топології графа та мітку класу $y \in \{0,1\}$, де 1 – дропер, 0 – легітимне ПЗ. Архітектура моделі складається з наступних компонентів: 2 шари GAT з 8 головами уваги кожен; активація LeakyReLU ($\alpha = 0.2$) між шарами; Dropout (рівень 0,6) для уникнення перенавчання та нормалізації по шарах; Readout layer – середнє/агреговане представлення всіх вузлів, фінальний fully-connected шар з softmax або sigmoid (залежно від функції втрат), функція втрат базується на крос-ентропії, а оптимізація виконується за допомогою Adam [14].

Дані для навчання збалансовані, включають як реальні легітимні програми (архіватори, текстові редактори, браузери), так і шкідливі зразки дроперів, відібрані з баз VirusShare, MalwareBazaar, а також власноруч зібрані вручну з API-журналів реального виконання. Загальний обсяг даних який було використано для навчання включав 5000 зразків (50% дропери, 50% легітимне ПЗ).

Навчена модель демонструє високу точність при класифікації навіть мутованих дроперів, які не мають сигнатурної подібності до базових варіантів. На відміну від класичних підходів, які використовують лише ознаки з файлу або послідовності байтів [8, 9], запропонований метод аналізує структурну динаміку поведінки, що робить його стійким до упаковки, інжекції, застосування шифрування чи поліморфізму.

Окрім того, важливою властивістю є інтерпретованість — на основі векторів уваги можна побудувати візуалізацію того, які саме API та їх поєднання стали визначальними для класифікації зразка як дропера. Це створює передумови для інтеграції моделі в системи кіберзахисту не лише як «чорну скриньку», а як експлейн-блок для аналізу ризиків [13].

Таким чином, побудована модель поєднує переваги глибокого навчання, структурного аналізу графів і динамічного спостереження за поведінкою, формуючи ефективний інструмент виявлення шкідливих дроперів, здатний до виявлення як відомих, так і невідомих загроз. Схематичне зображення процесу аналізу ПЗ представлене на рис. 1.

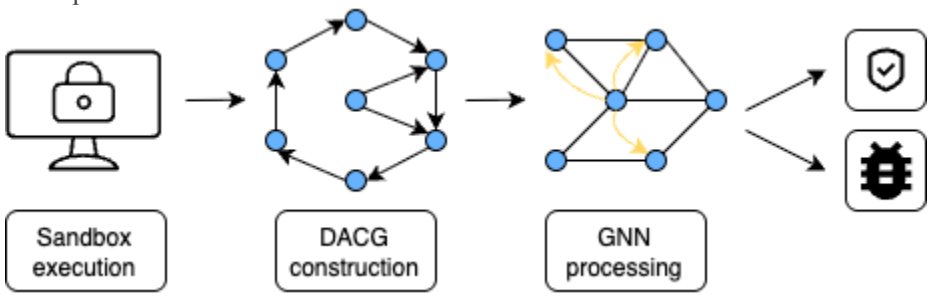

Рис. 1. Процес аналізу ПЗ

ЕКСПЕРИМЕНТ

Для оцінки ефективності запропонованого методу було проведено серію експериментів на різнорідних наборах даних, які містять як легітимне програмне забезпечення, так і зразки зловмисних дроперів. Джерелами тестових даних виступали відкриті репозиторії шкідливого ПЗ (зокрема, VirusShare, MalwareBazaar), а також виконувані журнали, згенеровані в результаті аналізу поведінки файлів у середовищі ANY.RUN та Cuckoo Sandbox. Особливу увагу було приділено формуванню збалансованого набору, що містив рівну кількість позитивних (дропери) та негативних (легітимні програми) прикладів, аби уникнути зміщення моделі під час навчання.

Кожен зразок попередньо виконувався в ізольованому середовищі з увімкненим динамічним трасуванням, після чого з отриманого API-журналу будувався граф поведінки відповідно до методології, викладеної в попередньому розділі. Загальна кількість оброблених зразків становила 5000: 2500 дроперів та 2500 легітимного ПЗ. Важливо зазначити, що в склад легітимного ПЗ включалися програми різних типів — офісні, архіватори, мережеві утиліти, браузери, системні компоненти — для відображення повного спектра реальних сценаріїв виконання.

Навчання моделі відбувалося на 70% цього набору, ще 15% використовувалися для валідації, а решта — для тестування. Модель GAT була реалізована з використанням фреймворку PyTorch Geometric. До її складу входили два шари графової уваги з 8 незалежними головами на кожному, активацією LeakyReLU та Dropout у розмірі 0.6. В якості оптимізатора було обрано Adam із коефіцієнтом навчання 0.005 та ваговим згладжуванням 5e-4. Модель тренувалася протягом 200 епох або до стабілізації функції втрат.

Метод протестовано на даних з EMBER 2018 та логах ANY.RUN, а саме оцінювання результатів здійснювалося за загальноприйнятими метриками якості класифікації. Було сформовано збалансований набір з 5000 зразків. Модель GAT порівнювалася з Random Forest, MLP та CNN. За метриками F1-score, AUC, точністю — запропонований підхід перевершив інші. Зокрема, GAT досягла F1 = 0.964, AUC = 0.986. Також виявлено високу стійкість до обфускації завдяки збереженню структурної поведінки у графі. Результати проведення результатів представленні у табл. 2.

Таблиця 2

Класифікація ефективності різних моделей у завданні виявлення дроперів

Модель	F1-score	Precision	Recall	AUC
GAT (наш метод)	0.964	0.958	0.971	0.986
Random Forest	0.894	0.876	0.913	0.926
MLP	0.882	0.853	0.910	0.901
CNN	0.907	0.890	0.925	0.936

Особливістю моделі GAT стала її здатність зберігати точність навіть на мутованих дроперах, які проходили перетворення за допомогою packer-утиліт (UPX, Themida), або містили вставки беззмістовного коду. Пояснюється це тим, що модель фокусувалася не на байт-коді або сигнатурах, а на реальній послідовності викликів API та логіці взаємодії з ОС. Навіть при зміні конкретних API, загальний структурний шаблон залишався характерним, що дозволяло мережі правильно ідентифікувати зразок як дропер.

Отримані результати підтверджують припущення про високу ефективність поєднання графових моделей із механізмами уваги у задачах виявлення динамічно змінюваних загроз. Запропонований підхід перевершує традиційні алгоритми як за точністю, так і за стійкістю до поліморфізму, а також демонструє добру узагальнювальну здатність при роботі з реальними зразками, які не входили до навчального набору.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Розроблено метод виявлення зловмисних дроперів у комп'ютерних системах, що ґрунтується на побудові орієнтованого графа викликів API та подальшій класифікації поведінкових шаблонів за допомогою графової нейронної мережі типу Graph Attention Network. На відміну від класичних сигнатурних та евристичних підходів, запропонований метод не залежить від конкретного вигляду коду або структури файлу, а фокусується на характерних закономірностях у послідовності та взаємозв'язках викликів системних функцій.

Проведені експерименти засвідчили високу ефективність запропонованого підходу: модель GAT продемонструвала точність 95.8%, повноту 97.1%, F1-міру 0.964 і AUC = 0.986, що суттєво перевищує показники базових моделей на основі MLP, Random Forest і CNN. Особливо цінним є те, що метод зберігає високу якість класифікації навіть у випадках із використанням обфускації, упаковки або метаморфізму, тобто у ситуаціях, де традиційні засоби виявлення зазнають невдачі.

Побудова графа викликів API дозволяє відобразити поведінку програми як структуровану мережу взаємодій, а механізм уваги GAT — виділити найбільш критичні ділянки такої поведінки, що істотно підвищує інтерпретованість і точність прийнятих рішень. У результаті запропонований підхід може бути застосований як окремо, так і як частина комплексної системи виявлення загроз у складі розподілених або хмарних архітектур кіберзахисту.

Подальші напрями досліджень включають:

розширення моделі за рахунок інтеграції мережевої активності та поведінки файлової системи у граф поведінки;

використання альтернативних графових архітектур, зокрема Graph Transformer або GraphSAGE, для покращення узагальнювальної здатності;

адаптацію моделі до потокової обробки журналів у реальному часі;

побудову інтерпретованих рішень з візуалізацією важливих API-шляхів для аналітиків SOC;

інтеграцію з розподіленими архітектурами, подібними до тих, що описані у роботах [1], [2], а також із системами на основі пасток та наживок [3], [5].

Отримані результати створюють основу для розробки нових підходів до поведінкового аналізу загроз, орієнтованих на структурну складність, динамічну активність та високий рівень узагальнення, необхідний для боротьби з сучасними дроперами та іншими складними шкідливими компонентами.

Література

1. A. Kashtalian, S. Lysenko, A. Sachenko, B. Savenko, O. Savenko, and A. Nicheporuk, “Evaluation criteria of centralization options in the architecture of multicomputer systems with traps and baits,” Radioelectronic and Computer Systems, vol. 2025, no. 1, pp. 264–297, 2025. https://doi.org/10.32620/reks.2025.1.18

2. A. Kashtalian, S. Lysenko, O. Savenko, A. Nicheporuk, T. Sochor, and V. Avsiyevych, “Multi-computer malware detection systems with metamorphic functionality,” Radioelectronic and Computer Systems, vol. 2024, no. 1, pp. 152–175, 2024. https://doi.org/10.32620/reks.2024.1.13

3. O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, and K. Bobrovnikova, “A technique for the botnet detection based on DNS-traffic analysis,” Communications in Computer and Information Science, vol. 522, pp. 127–138, 2015.

4. S. Lysenko, O. Pomorova, O. Savenko, A. Kryshchuk, and K. Bobrovnikova, “DNS-based anti-evasion technique for botnets detection,” in Proc. of the 8th IEEE IDAACS, Warsaw, Poland, pp. 453–458, 2015.

5 B. Savenko, S. Lysenko, K. Bobrovnikova, O. Savenko, and G. Markowsky, “Detection of DNS tunneling botnets,” in Proc. of the 11th IEEE IDAACS, Krakow, Poland, 2021.

6. D. Denysiuk, O. Savenko, S. Lysenko, B. Savenko, and A. Kashtalian, “Method for detecting steganographic changes in images using machine learning,” in Proc. of the 13th IEEE DESSERT, Athens, Greece, pp. 1–6, 2023. https://doi.org/10.1109/DESSERT61349.2023.10416453

7. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y., “Graph Attention Networks”, Art. no. arXiv:1710.10903, 2017. https://doi.org/10.48550/arXiv.1710.10903

8. D. Ucci, L. Aniello, and R. Baldoni, “Survey of machine learning techniques for malware analysis,” Computers & Security, vol. 81, pp. 123–147, 2019, ISSN 0167-4048, https://doi.org/10.1016/j.cose.2018.11.001

9. Jeremy Z. Kolter and Marcus A. Maloof. 2004. Learning to detect malicious executables in the wild. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '04). Association for Computing Machinery, New York, NY, USA, 470–478. https://doi.org/10.1145/1014052.1014105

10. M. Sikorski and A. Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. San Francisco, CA, USA: No Starch Press, 2012.

11. J. Zhang, S. Wang, and Y. Xue, “Towards fine-grained malware classification using API call graph embeddings,” in Proc. of IEEE BigData, pp. 2586–2595, 2019.

12. E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, “Malware detection by eating a whole EXE,” in Proc. of AAAI Workshops, 2018.

13. S. Brody, U. Alon, and E. Yahav, “How attentive are graph attention networks?,” arXiv preprint arXiv:2103.16143, 2021.

14. D. Gibert, C. Mateu, and J. Planes, “The rise of machine learning for detection and classification of malware: Research developments, trends and challenges,” Journal of Network and Computer Applications, vol. 153, 2020.

15. T. H. Le, D. T. Hoang, and D. Phung, “A survey on graph neural networks for cybersecurity,” ACM Computing Surveys, vol. 55, no. 9, pp. 1–36, 2022.

16. J. Saxe and K. Berlin, “Deep neural network-based malware detection using two-dimensional binary program features,” in Proc. of the 10th International Conference on Malicious and Unwanted Software (MALWARE), 2015.

17. M. Demontis, A. Melis, B. Biggio, and F. Roli, “Yes, machine learning can be more secure! A practical tutorial on adversarial machine learning,” in Proc. of ACM CCS Workshop, 2020.

18. S. Hou, A. Saas, Y. Chen, and C. M. Fung, “MalFox: A Behavioral Graph-Based Approach for Detecting and Classifying Malware,” in Proc. of RAID, 2021.

19. P. Vinayakumar, K. P. Soman, and P. Poornachandran, “Evaluating deep learning approaches to characterize and classify malware,” Computers & Security, vol. 70, pp. 92–107, 2017.

20. D. A. Berkowitz, “The AVClass Framework: Automatic Labeling of Malware Families,” IEEE Security & Privacy, vol. 17, no. 2, pp. 72–77, 2019.

21. J. Z. Kolter and E. Wong, “Provable defenses against adversarial examples via the convex outer adversarial polytope,” in Proc. of ICML, 2017.

22. A. David, M. V. Mahoney, and R. Chaki, “Survey of Machine Learning Algorithms for Malware Detection,” Journal of Computer Virology and Hacking Techniques, vol. 19, 2023.

23. J. W. Stokes and Y. Xie, “Detecting Malicious Web Downloads Using Large-Scale Graph Learning,” in Proc. of WWW, 2022.

24. S. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective Android malware detection using machine learning,” in Proc. of NDSS, 2014.

25. K. I. Wangsness, M. Abu Rajab, and R. Zhao, “Ghidra: NSA’s Reverse Engineering Platform in Practice,” IEEE S&P Mag., vol. 19, no. 3, pp. 34–42, 2021.

26 .F. Breitinger and S. Bursztein, “MalwareBazaar: A free database for malware research,” Digital Investigation, vol. 36, 2021.

27. S. K. Bashir and A. H. Mir, “A review on the use of deep learning for malware classification,” Artificial Intelligence Review, vol. 54, no. 5, pp. 3567–3614, 2021.

28. C. Yin, Y. Zhu, S. Liu, and J. Fei, “Deep learning for malware detection: Approaches, challenges and opportunities,” in Proc. of IEEE ICC, 2021.

29. R. Pascanu, T. Mikolov, and Y. Bengio, “Learning longer memory in recurrent neural networks,” arXiv preprint arXiv:1305.5770, 2013.

30. M. Vaswani, A. Jones, and I. Shapiro, “Malformer: Stacked transformer encoders for malware detection,” in Proc. of IEEE S&P Workshops, 2021.

31. A. Gupta et al., “Mamba for malware detection: Long-sequence attention with efficiency,” arXiv preprint arXiv:2311.07977, 2023.

32. Z. Chen, Y. Liu, and Q. Wang, “Attention-based CNN for behavioral malware classification,” Computers & Security, vol. 106, 2021.

33. W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in Proc. of NeurIPS, 2017. (GraphSAGE)

34. R. Ying et al., “Hierarchical graph representation learning with differentiable pooling,” in Proc. of NeurIPS, 2018. (DiffPool)

35. Y. Ying and J. You, “Are transformers good graph learners? A positional analysis,” in Proc. of ICLR, 2021. (Graph Transformer)

36. ISO/IEC 25010:2011, “Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models.”

37. J.-L. Letouzey, “The SQALE method for evaluating technical debt,” in Proc. of ICSME, 2014.

38. T. Mens and T. Tourwé, “A survey of software refactoring,” IEEE Trans. on Software Engineering, vol. 30, no. 2, pp. 126–139, 2004.

39. B. Kitchenham and S. Charters, “Towards an evidence-based approach in software engineering,” in Proc. of ICSE, 2007.

40. M. A. Storey et al., “How developers use metrics in software projects: A case study,” Empirical Software Engineering, vol. 23, no. 1, pp. 1–29, 2018.

41. Kashtalian, A., Ścisło, Ł., Rucki, R., Lysenko, S., Sachenko, A., Savenko, B., Savenko, O., & Nicheporuk, A. (2025). Control and Decision-Making in Deceptive Multi-Computer Systems Based on Previous Experience for Cybersecurity of Critical Infrastructure. *Applied Sciences*, *15*(22), 12286. https://doi.org/10.3390/app152212286

42. L. Bedratyuk, O. Savenko, *MATCH Commun. Math. Comput. Chem.* 2018, 79, 407–414.

43. B. Savenko, A. Kashtalian, S. Lysenko and O. Savenko, "Malware Detection By Distributed Systems with Partial Centralization," 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Dortmund, Germany, 2023, pp. 265-270, https://doi.org/10.1109/IDAACS58523.2023.10348773